Introduction to Python 2

Chang Y. Chung

Office of Population Research

May 2015
Algorithms + Data Structures = Programs

- Niklaus Wirth (1976)[?]

Python’s built-in data structures include:
- List
- Dictionary
- Tuple

We will also briefly talk about:
- Class
- Exception Handling
Algorithms + Data Structures = Programs

- Niklaus Wirth (1976) [1]
- Python’s built-in data structures include:
  - List
  - Dictionary
  - Tuple
Algorithms + Data Structures = Programs

- Niklaus Wirth (1976)
- Python’s built-in data structures include:
  - List
  - Dictionary
  - Tuple
- We will also briefly talk about:
  - Class
  - Exception Handling
List

- Ordered (indexed) collection of arbitrary objects.
- Mutable – may be changed in place.
List

Ordered collection of arbitrary objects.

```python
L = []  # a new empty list
L = list()  # ditto

L = [1, 2.5, "abc", [56.7, 78.9]]
print len(L)  # 4
print L[1]  # 2.5 (zero-based)
print L[3][0]  # 56.7

for x in L:
    print x
    # 1
    # 2.5
    # "abc"
    # [56.7, 78.9]

print "abc" in L, L.count("abc"), L.index("abc")  # True 1 2
```
List

- **Mutable** – may be changed in place.

```python
L = []
L.append(5)
print L  # [5]

L[0] = 23
print L  # [23]

M = [87, 999]
L.extend(M)  # or L += M
print L      # [23, 87, 999]

del L[2]
print L     # [23, 87]
```
More examples.

```python
def squares(a_list):
    s = []
    for el in a_list:
        s.append(el ** 2)
    return s

sq = squares([1,2,3,4])
print(sq, sum(sq))
# [1, 4, 9, 16] 30
```
More examples.

```python
def squares(a_list):
    s = []
    for el in a_list:
        s.append(el ** 2)
    return s

sq = squares([1, 2, 3, 4])
print sq, sum(sq)
# [1, 4, 9, 16] 30
```

Aliasing vs copying

```python
L = [1, 2, 3, 4]
M = L  # aliasing
L[0] = 87
print M  # [87, 2, 3, 4]

L = [1, 2, 3, 4]
M = list(L)  # (shallow) copying. M = L[:] also works
L[0] = 87
print M  # [1, 2, 3, 4]
```
Quiz

Given a list,

\[ L = [1, 2, [3, 4], 5, "xyz"] \]

evaluate the following expressions:

1. \( L[1] == 1 \)
2. \( \text{len}(L) == 5 \)
3. \( L[2] == 3, 4 \)
4. \( [3] \text{ in } L \)
5. \( L.index("xyz") == 4 \)
6. \( L[-1] == "xyz" \)
7. \( L[-1][-1] == "z" \)
8. \( \text{any}([1, 2, 3]) == \text{True} \)
9. \( L[9] == \text{None} \)
10. \( \text{len}([0,1,2,]) == 3 \)
Quiz

► Write a function that, given a list of integers, returns a new list of odd numbers only. For instance, given the list, [0,1,2,3,4], this function should return a new list, [1,3].
(Hint: Create a new empty list. Loop over the old one appending only odd numbers into the new one. Return the new one.)

```python
def only_odd(a_list):
    L = []
    for el in a_list:
        if el % 2 == 1:
            L.append(el)
    return L

print(only_odd([0, 1, 2, 3, 4]))  # [1, 3]
```
Write a function that, given a list of integers, returns a new list of odd numbers only. For instance, given the list, [0,1,2,3,4], this function should return a new list, [1,3]. (Hint: Create a new empty list. Loop over the old one appending only odd numbers into the new one. Return the new one.)

An answer.

```python
def only_odd(a_list):
    L = []
    for el in a_list:
        if el % 2 == 1:
            L.append(el)
    return L

print only_odd([0, 1, 2, 3, 4])
# [1, 3]
```
(tricky) Write a function similar to the previous one. This time, however, do not return a new list. Just modify the given list so that it has only the odd numbers. (Hint: \texttt{del L[0]} removes the first element of the list, \texttt{L})

```python
def odd_only(a_list):
    i = 0
    length = len(a_list)
    while i < length:
        if (n % 2) == 0:
            del a_list[i]
        i += 1
        length -= 1
    L = [0, 1, 2, 3, 4]  # check
odd_only(L)
print L
```
Slice index

- Applies to any sequence types, including `list`, `str`, `tuple`, . . . .
- Has three (optional) parts separated by a colon (`:`), `start : end : step`, indicating `start` through but not past `end`, by `step`; Indices point *in-between* the elements.

```
 ++++++++  
 | p | y | t | h | o | n |  
 ++++++++  
 0 1 2 3 4 5 6
--6 --5 --4 --3 --2 --1
```

- Examples:
  ```python
  L = ["p", "y", "t", "h", "o", "n"]
  print L[:2]  # ["p", "y"] first two
  print L[1:3]  # ["y", "t"]
  print L[0:5:2]  # ["p", "t", "o"]
  print L[-1]  # n the last element
  print L[:]  # ["p", "y", "t", "h", "o", "n"] a (shallow) copy
  print L[3:]  # ["h", "o", "n"]
  print L[-2:]  # ["o", "n"] last two
  print L[:::-1]  # ["n", "o", "h", "t", "y", "p"] reversed
  ```
Quiz

Suppose that you collect friendship network data among six children, each of whom we identify with a number: 0, 1, . . . , 5. The data are represented as a list of lists, where each element list represents the element child’s friends.

```
L = [[1, 2], [0, 2, 3], [0, 1], [1, 4, 5], [3, 5], [3]]
```

For instance, the kid 0 friends with the kids 1 and 2, since `L[0] == [1, 2]` Calculate the average number of friends the children have. (Hint: `len()` returns the list size.)
Quiz

Suppose that you collect friendship network data among six children, each of whom we identify with a number: 0, 1, \ldots, 5. The data are represented as a list of lists, where each element list represents the element child’s friends.

\[ L = [[1, 2], [0, 2, 3], [0, 1], [1, 4, 5], [3, 5], [3]] \]

For instance, the kid 0 friends with the kids 1 and 2, since \( L[0] == [1, 2] \) Calculate the average number of friends the children have. (Hint: \( \text{len()} \) returns the list size.)

An answer:

\[
\begin{align*}
\text{total} & = 0.0 & \# \text{ make total a float type} \\
\text{for } \text{el} \text{ in } L: \\
\text{ total } & += \text{len} (\text{el}) \\
\text{avg} & = \text{total} / \text{len}(L) \\
\text{print} \text{ avg} \\
\text{# 2.1666}
\end{align*}
\]
(tricky) Write a function to check if all the friendship choices are reciprocated. It should take a list like previous one and return either True or False. (Hint: You may want to use a utility function below.)

```python
def mutual(a_list, ego, alter):
    return alter in a_list[ego] and ego in a_list[alter]

def all_reciprocated(a_list):
    for ego in range(len(a_list)):
        alters = a_list[ego]
        for alter in alters:
            if not mutual(a_list, ego, alter):
                return False
    return True
```
List Comprehension

▶ A concise way to create a list. An example:

```python
[x for x in range(5) if x % 2 == 1]  # [1, 3]
```

▶ An equivalent code using the for loop:

```python
L = []
for x in range(5):
    if x % 2 == 1:
        L.append(x)  # [1, 3]
```

▶ More examples.

```python
[x - 5 for x in range(6)]  # [-5, -4, -3, -2, -1, 0]
[abs(x) for x in [-2, -1, 0, 1]]  # [2, 1, 0, 1]
[x for x in range(6) if x == x**2]  # [0, 1]
[1 for x in [87, 999, "xyz"]]]  # [1, 1, 1]
[x - y for x in range(2) for y in [7, 8]]  # [-7, -8, -6, -7]
```
Dictionary

- A collection of key-value pairs.
- Indexed by keys.
- Mutable.
Dictionary

- A collection of key-value pairs.
- Indexed by keys.
- Mutable.
- Also known as associative array, map, symbol table, ...
- Usually implemented as a hash table.
Dictionary

A collection of key-value pairs, indexed by keys.

```python
D = {}  # an empty dictionary. D=dict() also works
D[ "one" ] = 1  # "one": 1
D[ "two" ] = 2  # "one": 1, "two": 2
print D  # {"one": 1, "two": 2}

print D.keys()  # ["two", "one"] arbitrary order!
print "three" in D.keys()  # False. "three" in D also works

D = {"Apple": 116, "Big Mac": 550}

for key in ["Apple", "Orange", "Big Mac"]:  
    if key in D:
        value = D[key]
        print "{} has {} calories".format(key, value)
    else:
        print "{} is not found in the dictionary".format(key)
# Apple has 116 calories
# Orange is not found in the dictionary
# Big Mac has 550 calories
```
Dictionary

More Dictionary examples.

```python
D = {
    "China": 1350,
    "India": 1221,
    "US": 317
}
for key in D.keys():
    print "Pop of {0}: {1} mil".format(key, D[key])
# Pop of India: 1221 mil
# Pop of China: 1350 mil
# Pop of US: 317 mil

D = {
    [1,2]: 23
}
# TypeError: unhashable type: 'list'

D = {
    2: [2, 3],
    200: [3, 4],
    95: [4, 5]
}  # OK
print D[2]  # [2, 3]
print D[200]  # [3, 4]
```
SAT has three subsections: Critical Reading, Mathematics, and Writing. A result of taking an SAT exam is three scores.

```python
# data
SAT = {"cr":780, "m":790, "w":760}

# usage
print SAT[ "m" ]  # 790
```
A Data Structure

- SAT has three subsections: Critical Reading, Mathematics, and Writing. A result of taking an SAT exam is three scores.

```python
# data
SAT = {"cr":780, "m":790, "w":760}

# usage
print SAT[ "m" ]  # 790
```

- You can take SAT exams more than once.

```python
# data
SATs = [{"cr":780, "m":790, "w":760},
        {"cr":800, "m":740, "w":790}]

# usage
print SATs[0]  # {"cr":780, "m":790, "w":760}
print SATs[0][ "cr" ]  # 780
```
Hypothetical SAT data for two people: Jane and Mary.

```python
SAT = {
    "Jane": {
        "lastname": "Thompson",
        "test": [{"cr": 700, "m": 690, "w":710}],
    },
    "Mary": {
        "lastname": "Smith",
        "test": [{"cr": 780, "m": 790, "w":760},
                  {"cr": 800, "m": 740, "w":790}]
    }
}

print SAT["Jane"]
# {"test": [{"cr": 700, "m": 690, "w": 710}, "lastname": "Thompson"]}

print SAT["Jane"]["lastname"] # Thompson
print SAT["Jane"]["test"] # [{"cr": 700, "m": 690, "w":710}]
print SAT["Jane"]["test"][0] # {"cr": 700, "m": 690, "w": 710}
print SAT["Jane"]["test"][0]["cr"] # 700

mary1 = SAT["Mary"]["test"][1]
print mary1["cr"] # 800
```
Make a dictionary of 2012 SAT percentile ranks for the scores from 680 to 700 and for all three subsections. The full table is available at http://tinyurl.com/k38xve8. Given this dictionary, say \( D \), a lookup, \( D[680]["cr"] \) should be evaluated to 93.
Make a dictionary of 2012 SAT percentile ranks for the scores from 680 to 700 and for all three subsections. The full table is available at http://tinyurl.com/k38xve8. Given this dictionary, say \( D \), a lookup, \( D[680][\text{"cr"}] \) should be evaluated to 93.

An answer.

```python
D = {700: {"cr": 95, "m": 93, "w": 96},
    690: {"cr": 94, "m": 92, "w": 95},
    680: {"cr": 93, "m": 90, "w": 94}}

print D[680]["cr"] # 93
```
(tricky) Write a new dictionary \( DD \) such that we look up the subsection first and then the score. That is, \( DD["cr"][680] \) should be evaluated to 93. (Hint: Start with a dictionary below.):

\[
DD = \{
    "cr": \{700: 95, 690: 94, 680: 93\},
    "m": \{700: 93, 690: 92, 680: 90\},
    "w": \{700: 96, 690: 95, 680: 94\}\}
\]
Tuples

- A sequence of values separated by commas.
- Immutable.
- Often automatically *unpacked*.
Tuples

- A sequence of values separated by commas. Immutable.

```python
T = tuple()  # empty tuple. T = () works also
N = (1)      # not a tuple
T = (1, 2, "abc")  # a tuple (1, 2, "abc")
print T[0]   # 1
T[0] = 9     # TypeError. immutable
```

- Often automatically unpacked.

```python
T = (2, 3)  # a is 2, b is 3
a, b = T    # a and b swapped.
a, b = b, a
D = {"x": 23, "y": 46}
D.items()   # [('y', 46), ('x', 23)]
for k, v in D.items():
    print "%s ==> %d" % (k, v)  # y ==> 46
                                # x ==> 23
```
Class

- **class** defines a (user-defined) type, a grouping of some data (properties) and functions that work on the data (methods).

- An object is an *instance* of a type.

- **Examples:**
  - `int` is a type; 23 is an object.
  - `str` a type; "abc" an object.
  - "word document file" a type; "my_diary.docx" is an object
  - We have been using objects.
Examples of Built-in Types

▶ The `str` type has a bunch of methods.

```python
"abc".upper()  # ABC
"abc".find("c")  # 2
"abc".split("b")  # ['a', 'c']
```

▶ `open()` function returns a `file` object (representing an opened file).

```python
with open("test.txt", "w") as my_file:
    my_file.write("first line\n")
    my_file.write("second line\n")
    my_file.write("third line")

print type(my_file)  # <type "file">
print dir(my_file)  # properties and methods

my_file.write("something")  # error. I/O on closed file
```
Let’s create a bank account type.

class BankAccount:

    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

Usage examples.

my_account = BankAccount(100)
my_account.withdraw(5)
print my_account.balance # 95

your_account = BankAccount()
your_account.deposit(100)
your_account.deposit(10)
print your_account.balance # 110
Quiz

Implement a `Person` type(or class) which has three properties (`first_name`, `last_name`, and `birth_year`); and two methods: `full_name()` and `age()`. The `age()` method should take the current year as an argument. You may use the template below.

```python
class Person:
    def __init__(self, first, last, year):
        self.first_name = first_name
        self.last_name = last_name
        self.birth_year = birth_year

    def full_name(self):
        return self.first_name + " " + self.last_name

    def age(self, current_year):
        return current_year - self.birth_year

# check
mr_park = Person("Jae–sang", "Park", 1977)
print mr_park.full_name()  # Jae–sang Park
print mr_park.age(2014)    # 37
```
Inheritance

- A mechanism for code reuse in object-oriented programming (OOP).
- A subtype is a specialized basetype.

```python
import webbrowser

class CoolPerson(Person):
    def __init__(self, name, birth_year, video):
        Person.__init__(self, name, None, birth_year)
        self.video = video
    def full_name(self):
        return self.first_name
    def show_off(self):
        url = "http://www.youtube.com/watch?v={0}"
        webbrowser.open(url.format(self.video))

# check
psy = CoolPerson("PSY", 1977, "9bZkp7q19f0")
print psy.full_name()  # PSY
print psy.age(2012)  # 35
psy.show_off()  # show off the style
```
Exception Handling

- An exception is raised when a (run-time) error occurs. By default, the script stops running immediately.

```python
L = [0, 1, 2, 3]
print L[5]
# IndexError: list index out of range
```

- `try: ... except: ...` let us catch the exception and handle it.

```python
L = [0, 1, 2, 3]
try:
    print L[5]
except IndexError:
    print "no such element"
    print "next"
# no such element
# next
```
We can raise (or throw) an exception as well.

```python
1 def fetch(a_list, index):
2     if index >= len(a_list):
3         raise IndexError("Uh, oh!")
4     return a_list[index]
5
6 print fetch(L, 5)
7 # IndexError: Uh, oh!
```

Script can keep going if you catch and handle the exception.

```python
1 L = [0, 1, 2, 3]
2 try:
3     print fetch(L, 5)  # this raises an exception
4 except IndexError:
5     print "an exception occurred"
6     print "next"
7 # an exception occurred
8 # next
```
**An Example**

- `urlopen()` in `urllib2` module raises an exception when the web page is not found.

```python
import urllib2

L = ["http://google.com",
     "http://google.com/somethingfantastic",
     "http://yahoo.com"]

# we want to open each page in turn
for url in L:
    try:
        page = urllib2.urlopen(url)
        print page.getcode()
    except urllib2.HTTPError:
        print "failed to open: {0}".format(url)

# 200 (a return code of 200 means OK)
# failed to open: http://google.com/somethingfantastic
# 200
```
Summary

- Dictionary – A collection of key-value pairs. Mutable.
- Tuple – A sequence of values separated by commas. Immutable.
- Class – Defines a type, a grouping of properties and methods.
- try: ... except: ... – Catch and handle exceptions.